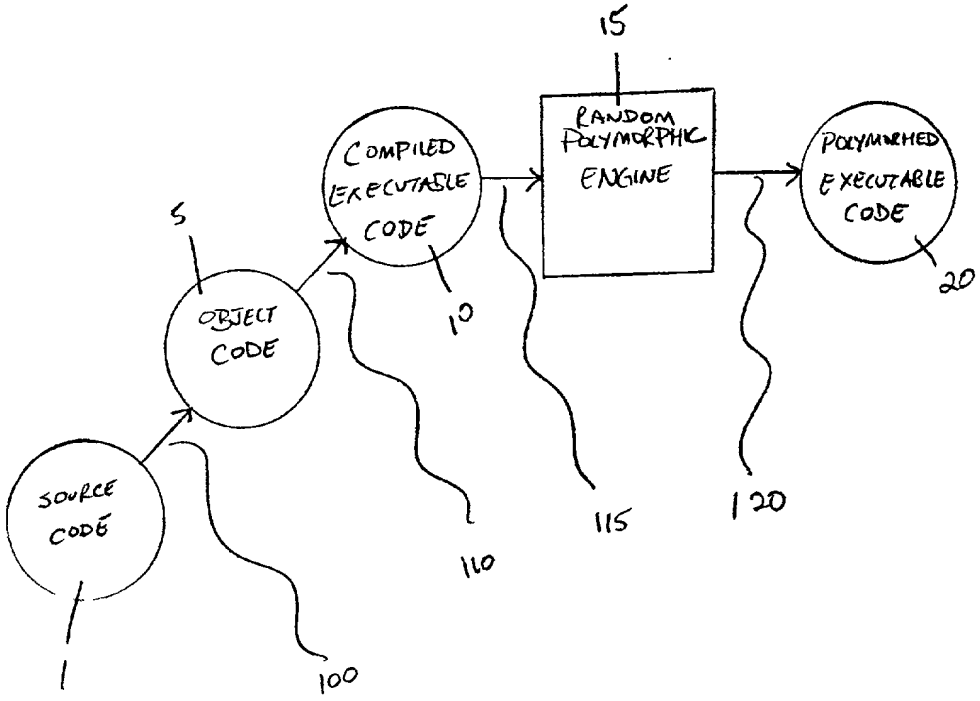

FIG. 1

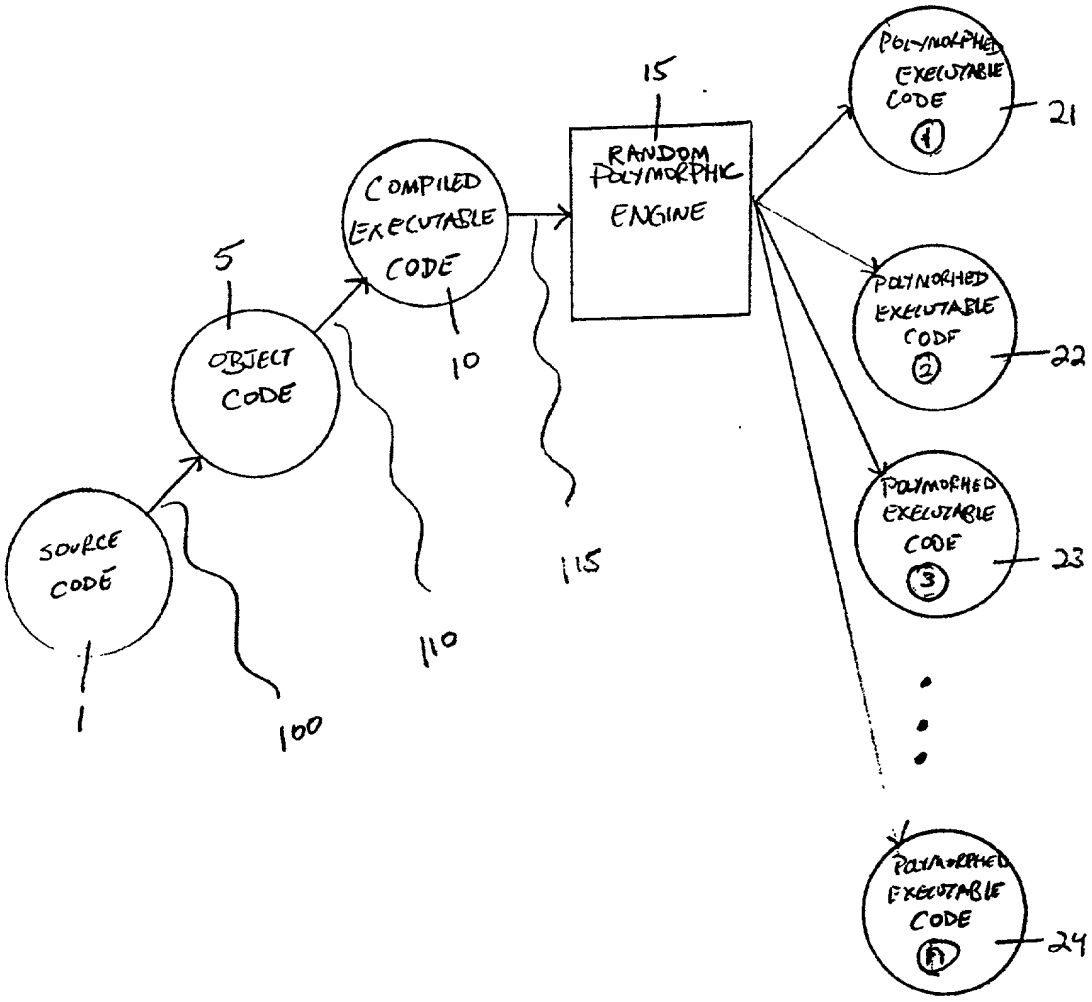
SOURCE CODE
1
OBJECT CODE
5
100
COMPILED EXECUTABLE CODE
10
110
115
RANDOM POLYMORPHIC ENGINE
15
POLYMORPHED EXECUTABLE CODE
1
21
POLYMORPHED EXECUTABLE CODE
2
22
POLYMORPHED EXECUTABLE CODE
3
23
POLYMORPHED EXECUTABLE CODE
N
24

FIG. 2

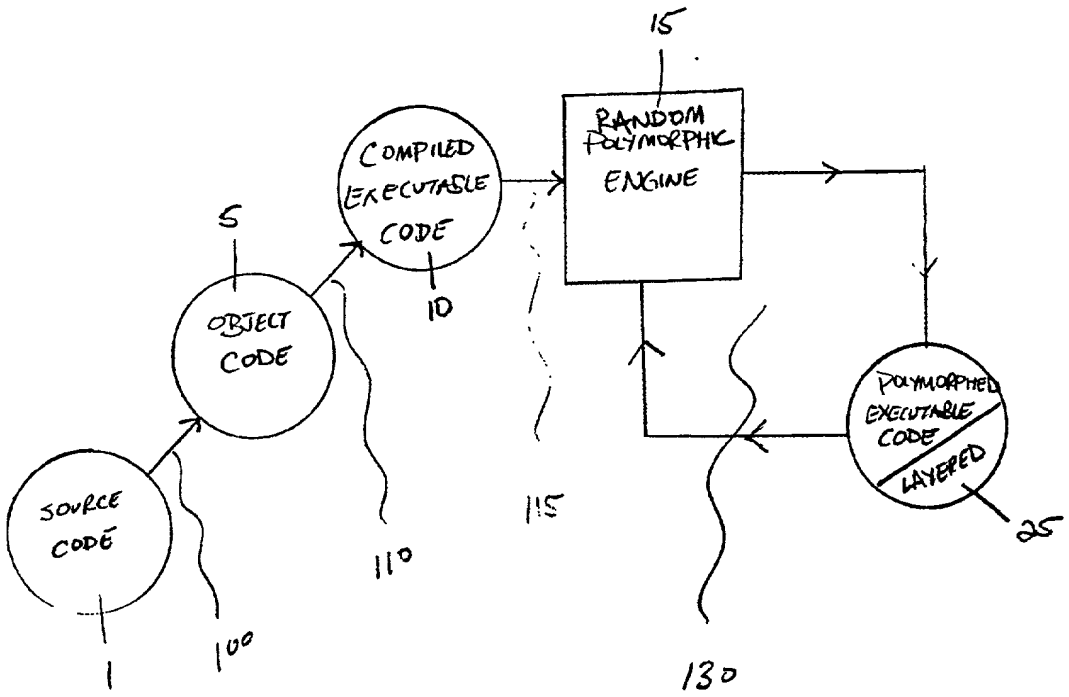
15
RANDOM POLYMORPHIC ENGINE
COMPILED EXECUTABLE CODE
10
5
OBJECT CODE
SOURCE CODE
1
100
110
115
130
POLYMORPHED EXECUTABLE CODE
LAYERED
25

FIG. 3

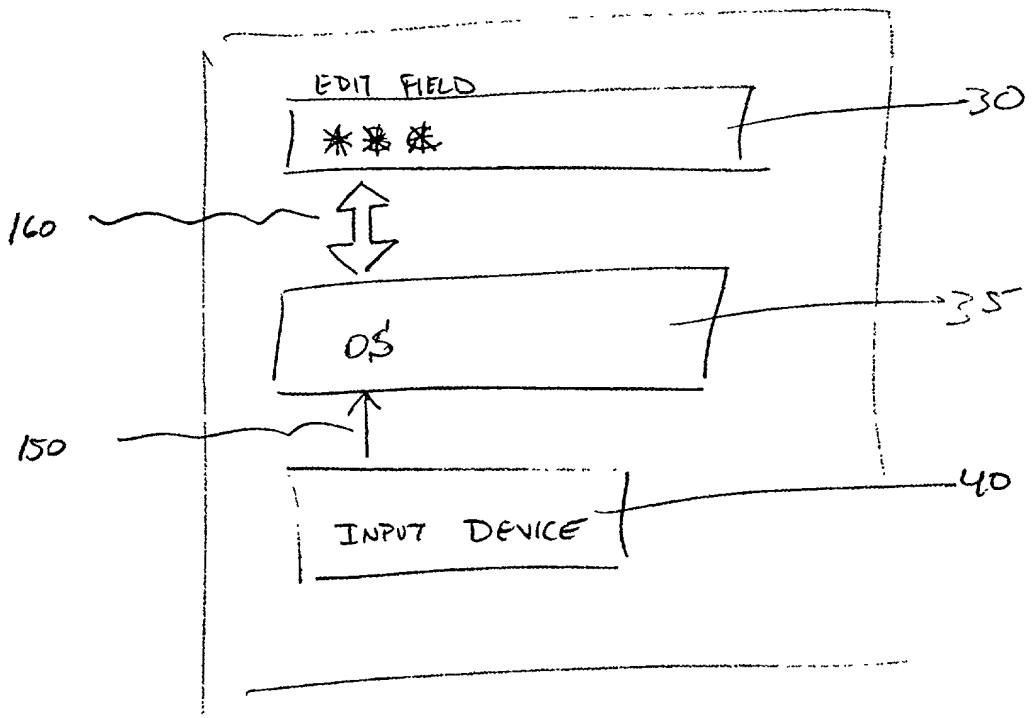

FIG. 4
PRIOR ART

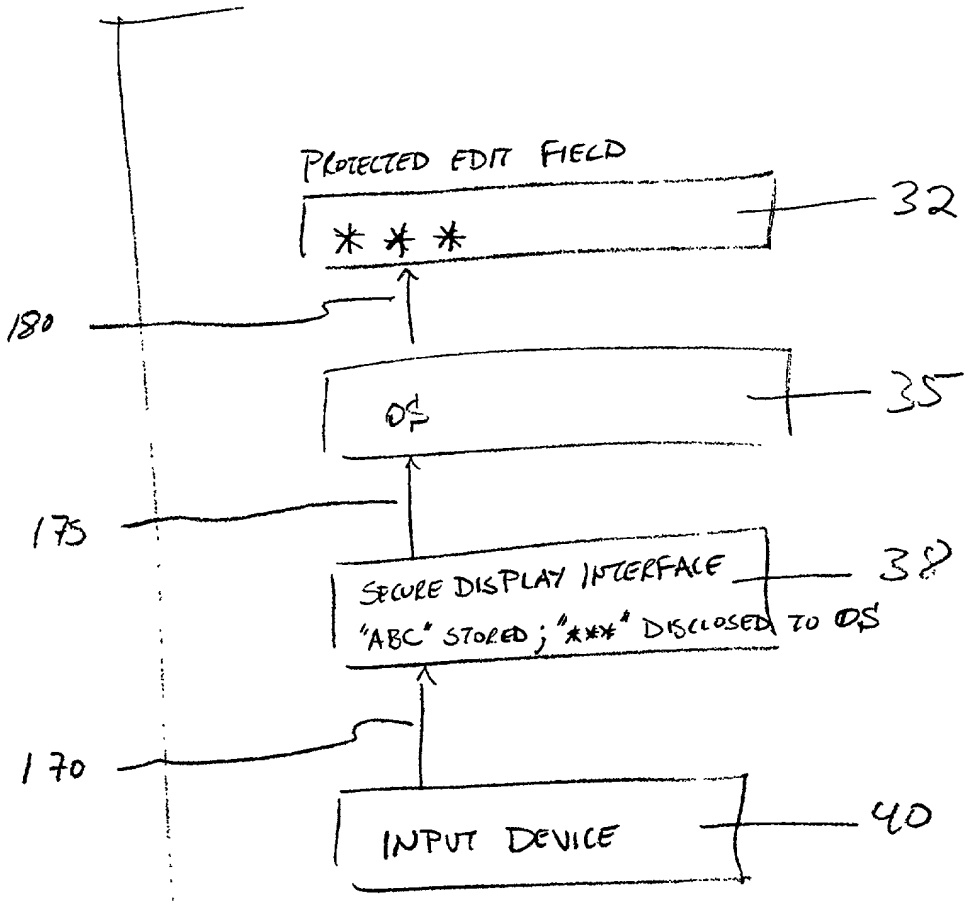
PROTECTED EDIT FIELD
***
32
180
OS
35
175
SECURE DISPLAY INTERFACE
"ABC" STORED; "***" DISCLOSED TO OS
38
170
INPUT DEVICE
40

FIG. 5

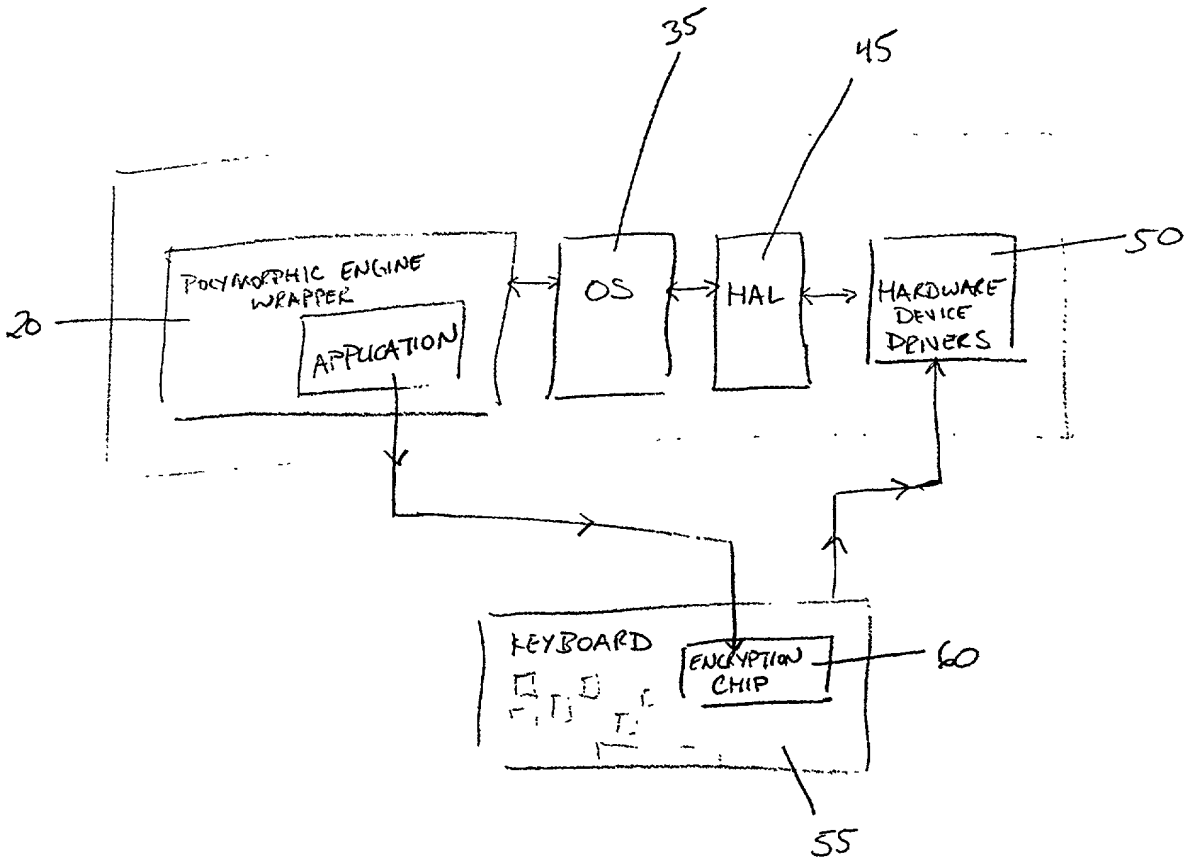
35
45
50
20
POLYMORPHIC ENGINE WRAPPER
APPLICATION
OS
HAL
HARDWARE DEVICE DRIVERS
KEYBOARD
ENCRYPTION CHIP
60
55

FIG. 6

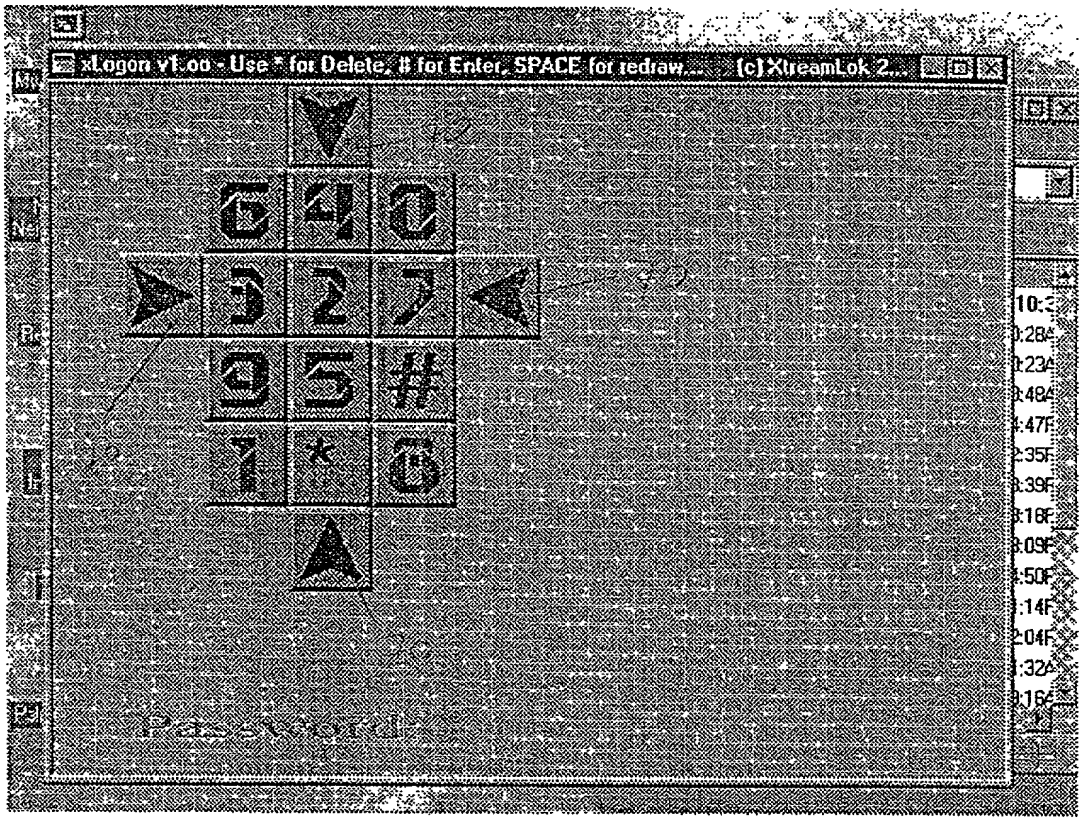
xLogon v1.00 - Use * for Delete, # for Enter, SPACE for redraw...
[c] XtreamLok 2...
Password:

FIG. 7

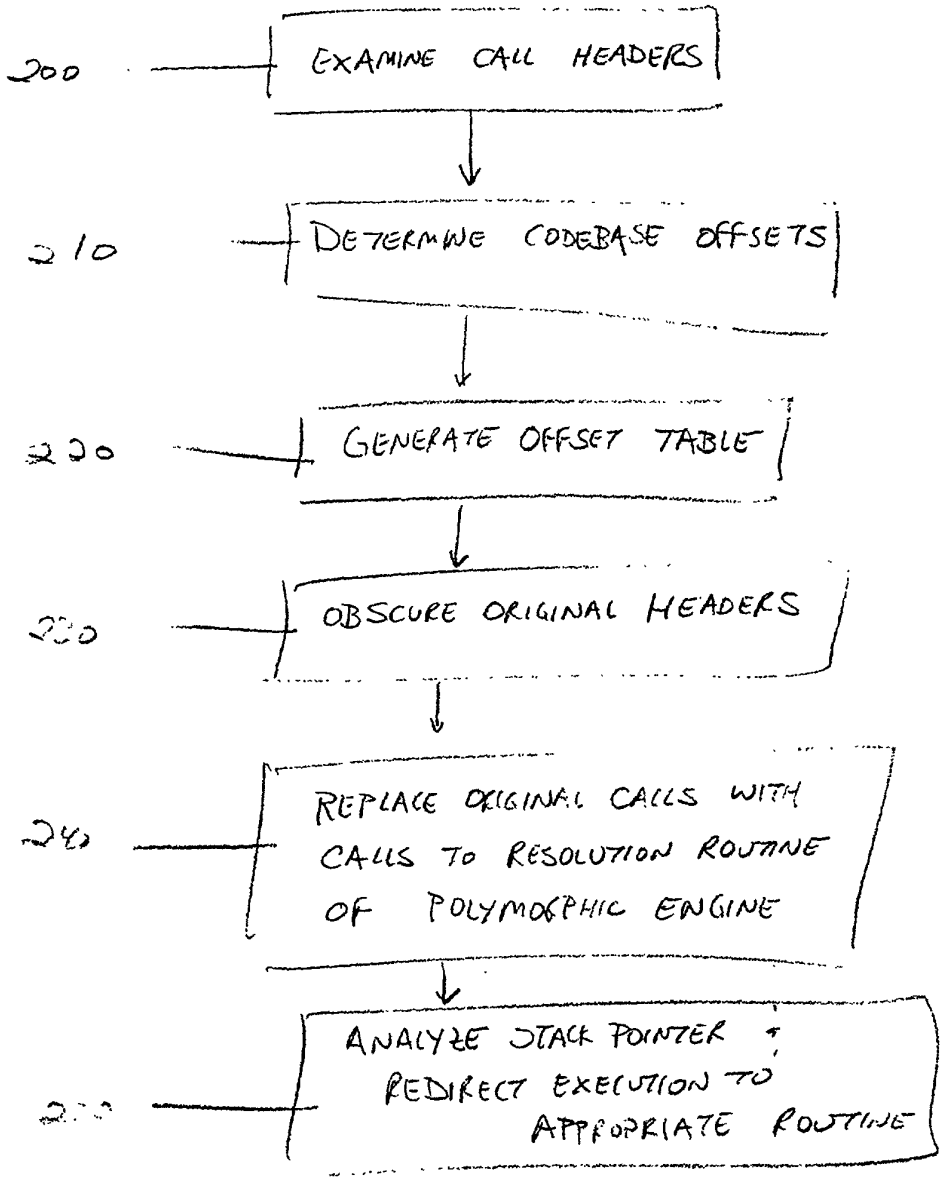

FIG. 8

Step 315	Checksum is calculated – modify any layer below this causes an invalid checksum to be calculated
Step 314	The polymorphic decryption engine is started using running line – modify any layer below or above this will cause the decryption to fail
Step 313	Validate the checksum calculated in **step 315** – any modification will cause a checksum error
Step 312	Erase the previous block – destroy above layers in memory, making rebuilding very tedious
Step 311	Fill internal import table – import table in the EXE is bogus; tamper with this layer and the EXE will not work
Step 310	Start running line with CRC, erase previous block, decode next block with CRC key – a single byte change in a layer above or below will destroy the next block of data
Step 309	Decrypt sections of running line – running line is based on the EXE CRC; modification of any layer will cause invalid data
Step 308	CRC PE header – store CRC for checking at a lower level
Step 307	Install timer code to trigger a CRC check
Step 306	Create import table – using layer at **step 311** to determine the actual import table to be used
Step 305	Decrypt resource sections – decrypt based on all layers above and below; any change will result in invalid data
Step 304	Erase previous; decrypt next – delete parts of the EXE as each block is decrypted, making a full rebuild very difficult
Step 303	Decode entry point and jump to it – decrypt the original entry point of the program based on all above layers; begin execution of the original program
Step 302	Check for breakpoint on each API call made and delete any hardware breakpoints – stops debugging
Step 301	Verify against modification every 'n' seconds – checks for debuggers; compares memory CRC with disk CRC from layer at **step 315**

FIG. 9

Original Instruction	Replacement Options
Functional Loop	Loop
	dec counter / jnz
	dec counter / jns
	dec neg[counter] / jz
	dec / cmp / jump far
Zero reg	xor reg,reg
	sub reg,reg
	mov reg, 0
MOV	mov reg,value
	zero_reg (see variations) + add reg, value
	zero_reg (see variations) + sub reg,value
load_reg – load reg 1 from [reg2]	mov reg, [Reg2]
	lods[b/w/d]
store_reg – stores reg1 to [reg2]	mov [reg2], reg1
	stos[b/w/d]
add / sub / inc / dec	add reg, value
	sub reg, neg value
	inc reg / conditional loop
	dec reg / conditional loop
Call / jmp	Push return address JMP address
	Call address

FIG. 10

Instruction (function required)	Actual Instruction used
Mov	Randomly selected from Fig. 10
Add	Randomly selected from Fig. 10
Sub	Randomly selected from Fig. 10
Xor	Use as is
Cmp	Use as is
Inc	Randomly selected from Fig. 10
Dec	Randomly selected from Fig. 10
Mov to register	Randomly selected from Fig. 10
Add to register	Randomly selected from Fig. 10
Sub to register	Randomly selected from Fig. 10
Xor to register	Use as is
Cmp to register	Use as is
Move register to register	Randomly selected from Fig. 10
Cmp register to register	Randomly selected from Fig. 10
Add register to register	Randomly selected from Fig. 10
Sub register to register	Use as is
One	Use as is
Jmp	Randomly selected from Fig. 10
Call	Randomly selected from Fig. 10
Or	Use as is
And	Use as is
Test	Use as is

FIG. 11

Line	Original Opcode	Mnemonics	Encrypted Opcode*	Encrypted Mnemonics	Runtime Decryption	
					Opcode	Mnemonics
1	8B 06	MOV EAX, DWORD PTR [ESI]	98 06	CWDE; PUSH ES	98 06	CWDE; PUSH ES
2	***83 F8 00***	***CMP EAX, 0***	***90 F8 00***	***NOP CLC***	***83 F8 00***	***CMP EAX, 0***
3	74 03	JZ LOC_1	67 03	ADD EDI+3,AH	67 03	ADD EDI+3,AH

*Key = 0x13

FIG. 12

EIP	Index
1000	0
14A9	1
1A02	2
280C	3
3A10	1
...	...

FIG. 13

Index	API
0	MessageBoxA*
1	CreateWindowExA
2	ExitProcess
3	WriteFile
. . .	. . .

* These a place holders not "real" text. Also they are encrypted in this table, not plain as shown here.

FIG. 14

EIP	API Call	Mutated API Call
0x1000	CALL [MessageBoxA]	CALL [polymorph_engine]
. . .	. . .	. . .
0x14A9	CALL [CreateWindowExA]	CALL [polymorph_engine]
. . .	. . .	. . .
0x1A02	JMP [ExitProcess]	CALL [polymorph_engine]
	PUSH EBX	
	CALL EAX	
. . .	. . .	. . .
0x280C	MOV EBP, [WriteFile]	CALL [polymorph_engine]
	CALL EBP	
	. . .	
	CALL EBP	
	. . .	
0x3A10	CALL [CreateWindowEXA]	CALL [polymorph_engine]

FIG. 15

Original Code	Opcode	Replacement Code	Replacement Opcode
Call dword ptr [API]	FF 15 xx xx xx xx	Call dword ptr [polymorph_engine]	FF 25 xx xx xx xx
Jmp dword ptr [API]	FF 25 xx xx xx xx	Call dword ptr [polymorph_engine]	FF 25 xx xx xx xx
Mov eax, dword ptr [API]	A1 xx xx xx xx	Call polymorph_engine	E8 xx xx xx xx
Mov eax, dword ptr [API]	8B 05 xx xx xx xx	Call polymorph_engine DB X1	E8 xx xx xx xx X1
Mov ebx, dword ptr [API]	8B 1D xx xx xx xx	Call polymorph_engine DB X2	E8 xx xx xx xx X2
Mov ecx, dword ptr [API]	8B 0D xx xx xx xx	Call polymorph_engine DB X3	E8 xx xx xx xx X3
Mov edx, dword ptr [API]	8B 15 xx xx xx xx	Call polymorph_engine DB X4	E8 xx xx xx xx X4
Mov edi, dword ptr [API]	8B 3D xx xx xx xx	Call polymorph_engine DB X5	E8 xx xx xx xx X5
Mov esi, dword ptr [API]	8B 35 xx xx xx xx	Call polymorph_engine DB X6	E8 xx xx xx xx X6
Mov esp, dword ptr [API]	8B 25 xx xx xx xx	Call polymorph_engine DB X7	E8 xx xx xx xx X7
Mov ebp, dword ptr [API]	8B 2D xx xx xx xx	Call polymorph_engine DB X8	E8 xx xx xx xx X8

X1 – 8 can be any of the following: EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP.

FIG. 16

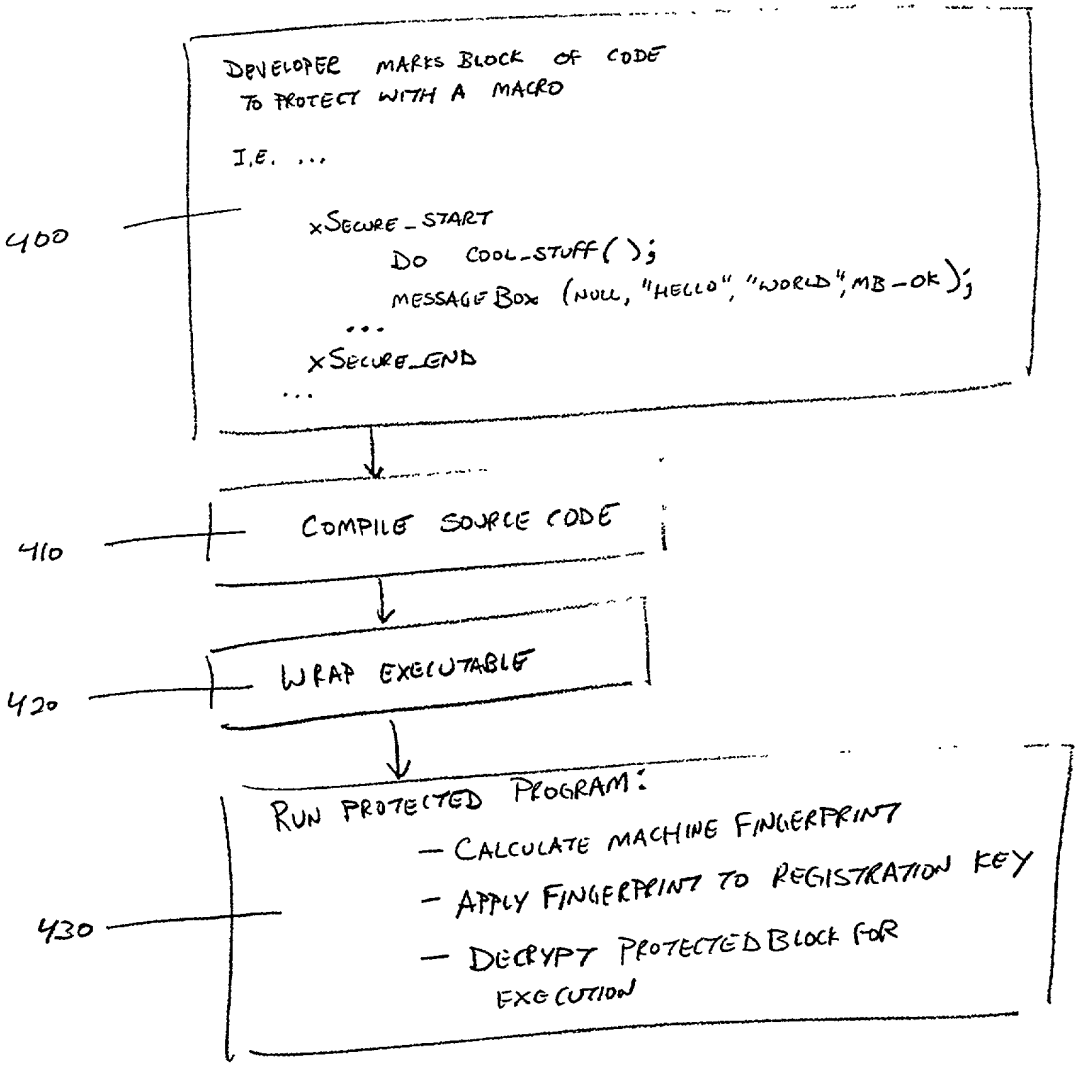
DEVELOPER MARKS BLOCK OF CODE TO PROTECT WITH A MACRO
I.E. ...
xSECURE_START
DO COOL_STUFF();
MESSAGEBOX (NULL, "HELLO", "WORLD", MB_OK);
...
xSECURE_END
...
400
COMPILE SOURCE CODE
410
WRAP EXECUTABLE
420
RUN PROTECTED PROGRAM:
— CALCULATE MACHINE FINGERPRINT
— APPLY FINGERPRINT TO REGISTRATION KEY
— DECRYPT PROTECTED BLOCK FOR EXECUTION
430

FIG. 17

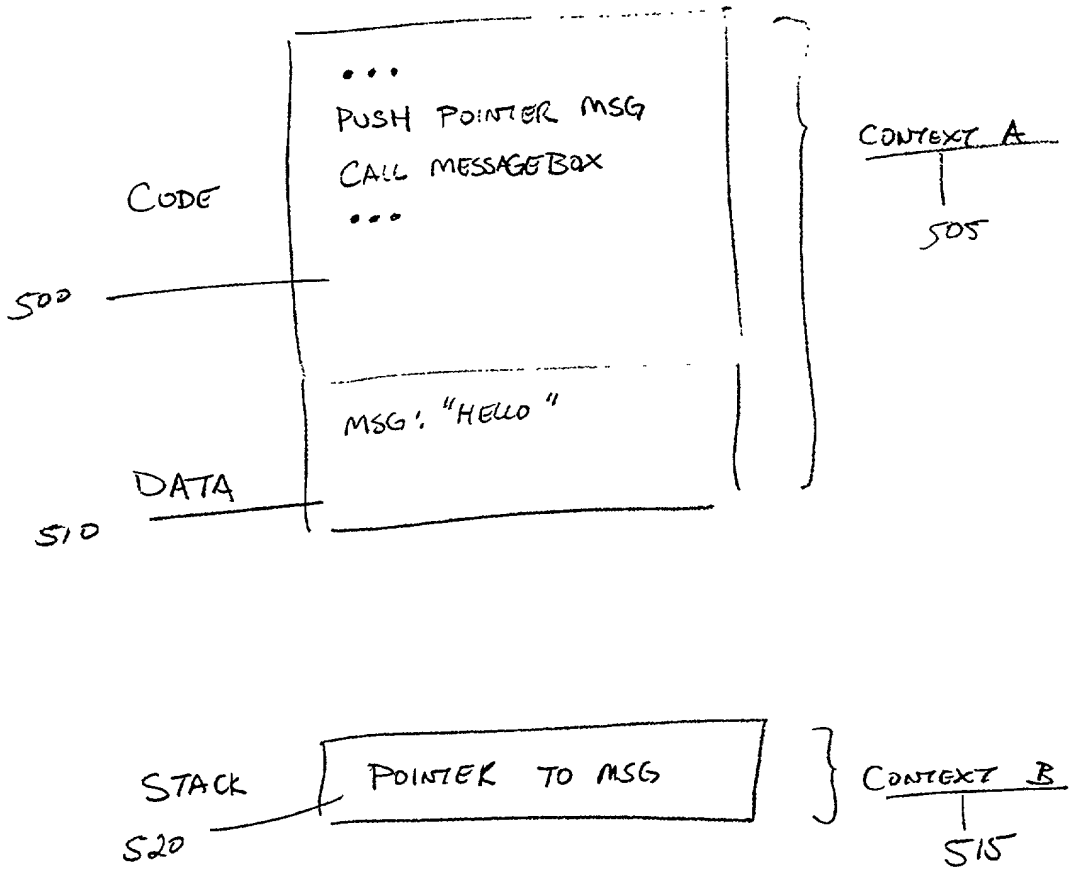

FIG. 18
PRIOR ART

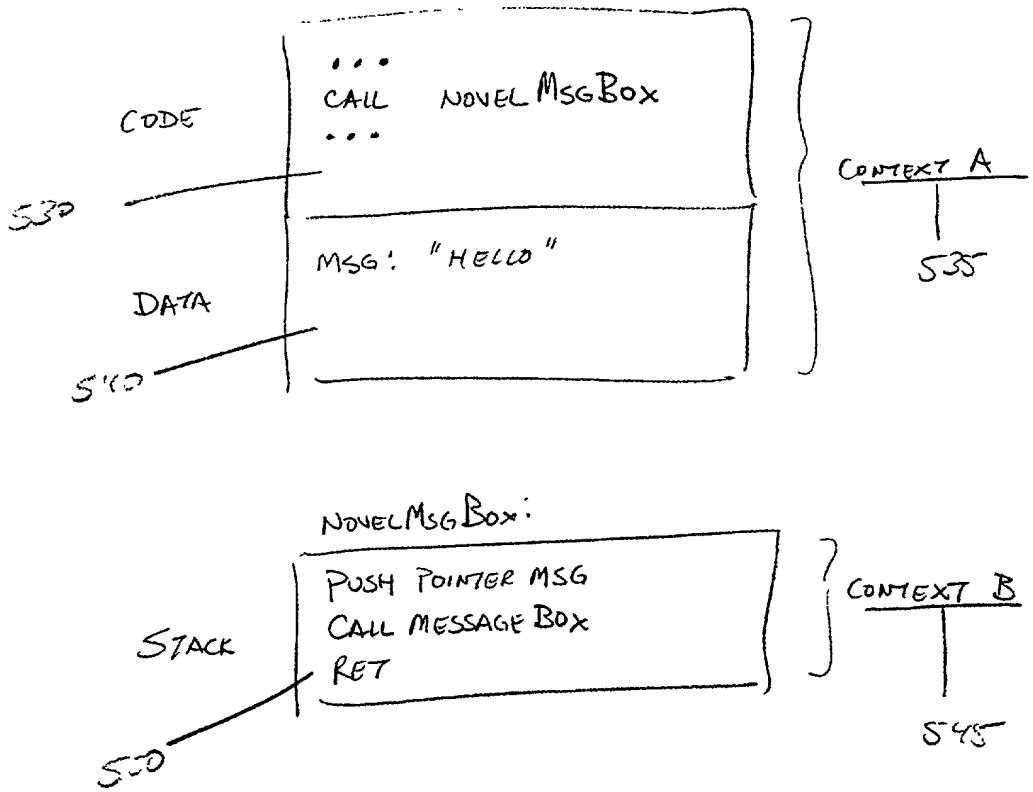
CODE
530
...
CALL NOVEL MSGBOX
...
MSG: "HELLO"
DATA
510
CONTEXT A
535
NOVELMSGBOX:
PUSH POINTER MSG
CALL MESSAGE BOX
RET
STACK
520
CONTEXT B
545

FIG. 19

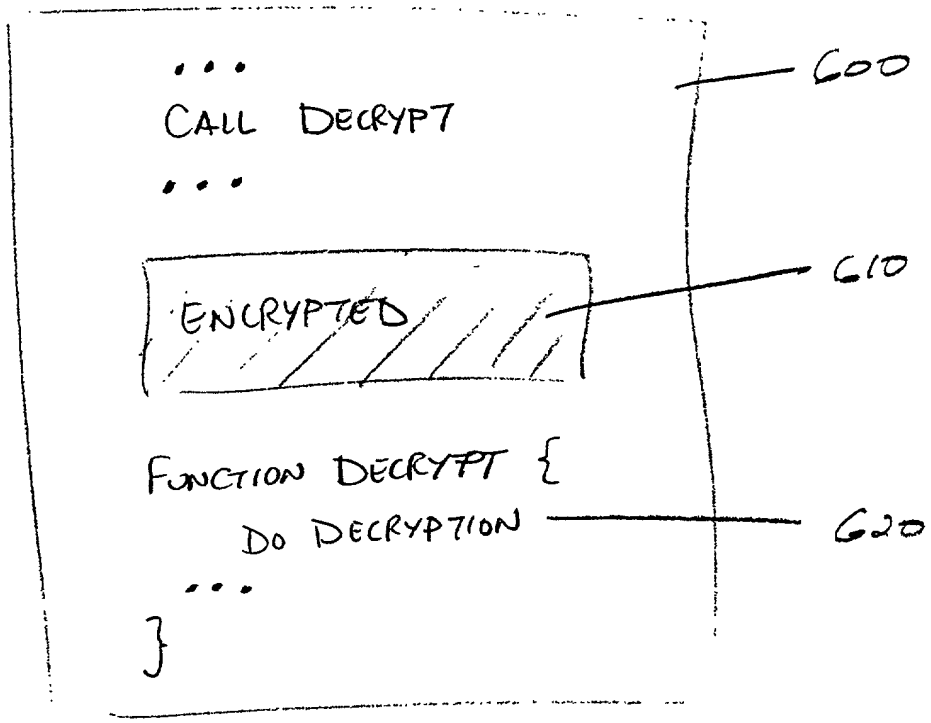

FIG. 20
PRIOR ART

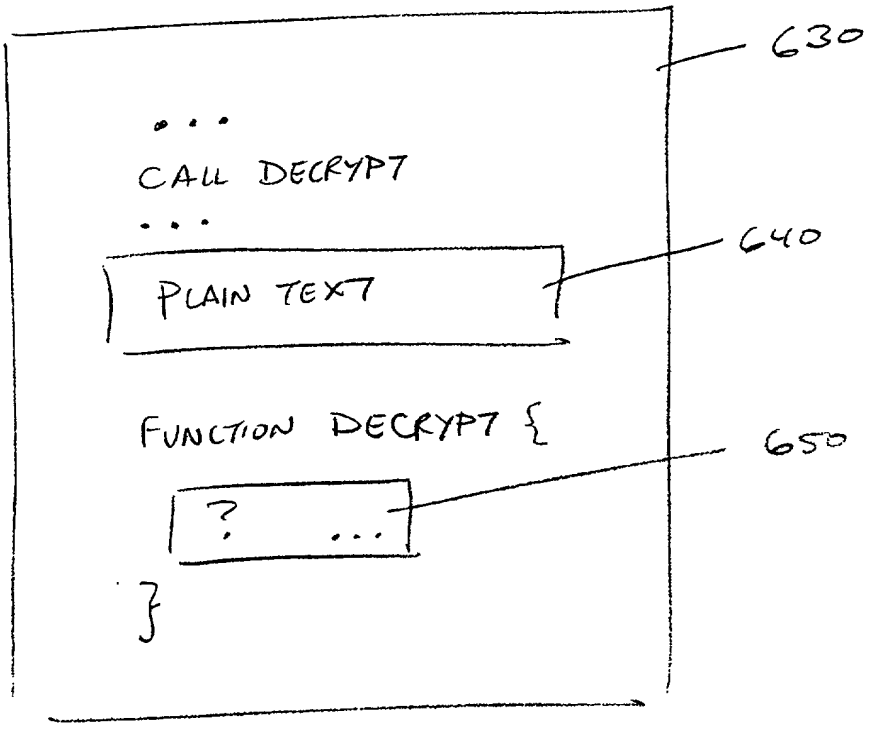

FIG. 21